Integrating Groupware Activities into Workflow Management Systems

Israel Z. Ben-Shaul
Department of Electrical Engineering
Technion-Israel Institute of Technology
Technion City, Haifa 32000
ISRAEL
issy@ee.technion.ac.il

Gail E. Kaiser
Department of Computer Science
Columbia University
New York, NY 10027
UNITED STATES
kaiser@cs.columbia.edu

Abstract

Computer supported cooperative work (CSCW) has been recognized as a crucial enabling technology for multi-user computer-based systems, particularly in cases where synchronous human-human interaction is required between geographically dispersed users. Workflow is an emerging technology that supports complex business processes in modern corporations by allowing to explicitly define the process, and by supporting its execution in a workflow management system (WFMS). Since workflow inherently involves humans carrying out parts of the process, it is only natural to explore how to synergize these two technologies. We analyze the relationships between groupware and workflow management, present our general approach to integrating synchronous groupware tools into a WFMS, and conclude with an example process that was implemented in the Oz *WFMS and integrated such tools. Our main contribution lies in the integration and synchronization of individual groupware activities into modeled workflow processes, as opposed to being a built-in part of the workflow WFMS.*

1. Introduction

Human-to-human collaboration and coordination is critical in any multi-person product development effort. In cases where the work requires intensive use of computers, computerized support for collaboration is essential, particularly when the collaborating users are physically dispersed, a scenario that is becoming more common with the recent advances in networking technologies and the growing popularity of the Internet and the World Wide Web.

Workflow management is an emerging technology that is concerned with modeling and executing *business processes*. As defined in the workflow coalition model [17], a business process is "a procedure where documents, information or tasks are passed between participants according to defined sets of rules to achieve, or contribute to, an overall business goal". A WorkFlow Management System (WFMS) thus provides a formalism (e.g., Petri nets, task-graphs) in which processes are defined, and a corresponding workflow engine in which processes are "executed", where forms of execution include automation in scheduling and activating activities according to the defined process; reactively triggering activities based on state changes; monitoring the process; and enforcing policies and consistency constraints (e.g., on the data being accessed during the process).

Since workflow inherently involves multiple humans carrying out parts of the process, it is only natural to explore how to synergize groupware and WFMS. This paper investigates support for defining groupware activities in a process, and executing them as part of an on-going workflow. Our focus is on integrating individual (external) synchronous tools, such as multi-user editors (e.g., Flecse [6]) and virtual whiteboards [18], into a process executed in a WFMS. This is in contrast to interfacing the WFMS framework with entire CSCW development toolkits or environment frameworks (e.g., ConversationBuilder [20]).

The possible degree of integration of groupware tools into the WFMS lies in a wide spectrum. The simplest method involves "inserting" a single tool as an isolated entity in the process and invoking it using the same notations and mechanisms as for regular (single-user) tools. This is obviously a very limited form of integration; for example, it does not supply mechanisms to identify and bind the participants to the execution of the activity from within the workflow framework, and it does not allow to associate the activity with other related activities. At the other end of the spectrum, a groupware activity may be fully integrated in the WFMS by becoming an undistinguished part of the WFMS framework *itself*, as opposed to being part of a particular process that is enacted on the WFMS. Such an approach is taken by various existing WFMSs (e.g., Lotus Notes [21]), in which the WFMS is essentially treated as a CSCW system. While useful in its own right, it does not address the need to integrate external tools which are defined as part of a specific process description, and for which the WFMS does not have its own "native support". This leads to our approach, which is process-specific tailoring and integration of groupware activities. This approach requires means to embed groupware tools such that it is possible to define control-flow, constraints and other rules of invocation for the activity, and to supply additional notations and mechanisms for handling multi-user synchronous interactions. Here again, there are several levels of integration of the tool with the underlying framework, ranging from being black-box where the internals of the tool (e.g., source code) are not accessible to the WFMS, to "gray-box" integration if the tool provides some application programming interface, to "white-box" integration. While white-box integration may have a potential for higher-degree of integration, it cannot be employed when there is no access to the tool's source code, or when modification of the (external) tool might be too difficult. Hence, our focus in this paper is on models and mechanisms that provide for full integration of individual groupware activities as units of a workflow process, but treating the tools themselves as encapsulated entities.

1.1. A Motivating Example

Consider a workflow for reviewing documents (e.g., research paper, or a business plan) by a group of independent and physically-dispersed experts. A `Review` task, illustrated in Figure 1, may be defined as follows. A coordinator sets-up the team of reviewers, by possibly running a `setup-review` tool that identifies and selects a qualified set of experts who agree to review the document. Next, they review the document individually. Once they all complete, a multi-user virtual conference meeting takes place, where they discuss together the document and possibly reconcile their conflicts. After the meeting, if the document is accepted, the coordinator completes the review task and proceeds with the approval task. Otherwise, if the document needs revisions, a revision request is sent to the author(s) of the document, after which the task reiterates to the personal review phase. If the review concludes that the document is totally unacceptable in its present form, it is rejected and a new submission (perhaps from another person) is requested. Each of these activities may be associated with an automated set of actions, or enforcement rules. For example, the `approve` action for paper review may entail notifying the author and the publisher and sending the proper author kit (or proceeding with actions to deliver the requested venture capital in case of a buisness plan review).

In order to support such task, the WFMS should have the ability to: (1) *define* such task using the WFMS process modeling formalism, including definition of the groupware activities, local and global constraints on their activation, and local and global inter-activity control-flow and "user"-flow; and (2) *execute* such task, including mechanisms that automate the control- and user-flow, enforce the constraints, and in general enabling groupware activities to affect, and be affected by, other local or groupware activities from same or from different users.

The rest of this paper is organized as follows: Section 2 discusses the relationships between groupware and workflow, and overviews related work; Section 3 presents our approach to enabling integration of groupware activities in a process; Section 4 outlines the realization of the approach in the Oz WFMS and presents an example Oz (sub) process that integrates groupware, and Section 5 summarizes the paper.

2. Relationships between Workflow and Groupware

Although both fields deal with certain common issues and overlap with each other, their orientation is quite different, and it is important to realize these differences as a basis for understanding both the need for, and the general approach to, our synergy. Workflow management in general focuses on support for *process*, including its explicit representation and executability, involving users, tools and artifacts. Further, it is concerned with allowing to specify and preserve the consistency and integrity of the process and its related artifacts (e.g., documents, products, etc.), particularly for tasks that require such support. Finally, it typically involves integration of single-user, or asynchronous multi-user tools (in fact, the WFMS may itself be viewed as such "tool").

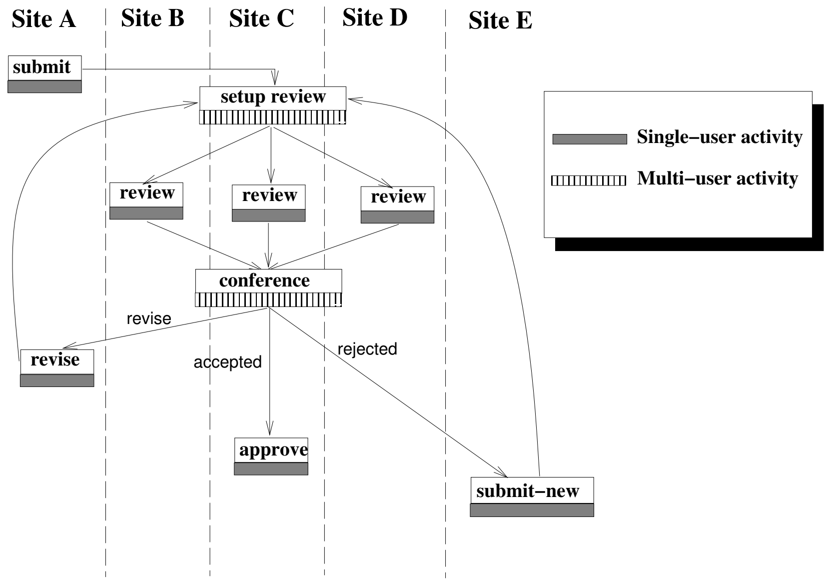

Figure 1: The Review Task

CSCW, to the contrary, is less concerned with (formal) processes, and is mainly concerned with human-human interface and inter-personal cooperation and collaboration paradigms. Process is incidental and implicit, and therefore unsupported. Thus, while both technologies are geared towards supporting collaboration among people in organizations, they are mostly complementary. Workflow technology can benefit from integrating groupware by embracing the human-human interaction paradigms and tools, and groupware could benefit from workflow by adding explicit and consistent process definition and enactment.

Although groupware in general refers also to asynchronous tools such as electronic mail and electronic bulletin-boards, in this paper we focus on the classical *synchronous* groupware tools, in which multiple users collaborate in a virtual shared space, also known as "same time, same place" technology [13]. Each such tool is invoked for or otherwise affiliated with a designated set of users all at the same time and they all terminate their connection at the same time (modulo network delays and other implementation-specific glitches). We have in mind tools ranging from multi-user editors and debuggers (e.g., the Flecse toolkit [6]) to document inspection systems (e.g., Scrutiny [11]). Most significantly from the viewpoint of workflow, synchronous groupware requires special integration facilities that do not exist in conventional WFMSs, and at the same time, once they are in place, there are many automation opportunities and control dependencies that can be associated with their activation, which is the main reason for focusing on them. (An asynchronous multi-user tool enveloping approach that allows the WFMS to submit multiple activities to the same "persistent" tool invocation is described elsewhere [24].)

There have been some systems that attempted to combine workflow and synchronous groupware. One such system that originated from CSCW is Conversation Builder (CB) [20]. The main concept in CB is that of a *conversation*, that serves as a context in which a user performs its actions ("utterances"), and can potentially affect other users participating in the same conversation through a shared conversation space yet still protect their private conversation space. In addition, CB enables to specify activities and their interrelations using *protocols*, which are state-machine descriptions of the flow of the conversations, or in other words, a limited form of process modeling. However, it has no process enactment engine.

Scrutiny [11] is a code-inspection system built on top of CB that supports a specific methodology — Fagan-style code inspection [7] — translated into process. This includes support for different roles (moderator, author, etc.) and ensuring that the inspection follows the defined process, in addition to the underlying groupware framework for supporting synchronous inspection among multiple remote users. Scrutiny is not a generic WFMS, however; the workflow process is built-in.

On the WFMS side, most systems provide some degree of "groupware", or multi-user support because they are inherently multi-user. However for the most part the support is either built into the WFMS (as opposed to integrating external tools), or it supports asynchronous external groupware. Examples include InConcert [5], ActionWorkflow [9] and ProcessWEAVER [8].

Finally, there is yet another approach to supporting collaboration that is orthogonal to both workflow and groupware, namely (collaborative) concurrency control. Users going about their own business happen to try to access the same data in conflicting ways (e.g., two writers). The concurrency control mechanism *reacts* to such attempts, and typically disallows all except one of the accesses. We have conducted extensive research on semantics-based concurrency control, as have others, and there are many proposed approaches that provide collaboration by permitting "conflicting" accesses when they happen to come up, perhaps with an attached "resolution" procedure (see [1] for a survey). We take here a complementary *proactive* approach to supporting collaboration.

3. Integration Concepts and Mechanisms

3.1. User Modeling

In order to identify and specify explicitly which users should be assigned to the execution of an activity, users must be somehow represented in the system. The simplest way to represent users is by their operating-system (or any other system-supplied) id. This allows the process to associate users with activities, either by directly specifying the id, or indirectly via a "user" attribute that is dynamically bound to such id at runtime. We defer the discussion of static vs. dynamic binding of users to activities to Section 3.2; regarding the representation, this approach is clearly limited. For example, it would be impossible to associate users to activities based on general characteristics and specific "state" of the users, such as their roles, whether they are active in the system, their physical and virtual location, etc. Moreover, the lack of such attributes may not allow the workflow engine to assist in the activation of groupware tools by, for example, locating available and qualified users.

Thus, a more suitable representation of users should be provided. Our approach is to model users as a distinct entity, much in the same way that typical WFMSs model data, process, and tools. In other words, in addition to data-, tool- and process- (or control-) integration, we employ user-integration. This leads to abstracting users as objects in a user-repository. More specifically, each user is represented as an object that stores pertinent information that is needed by the operating process, and is instantiated from a primitive *user* "base" class, or one of its derived sub-classes. Individual user objects can be aggregated in a *user-group*, which can be used for three different purposes: (1) represent an organizationally determined sets of users, such as members of a project team (an "is-part-of" composition hierarchy); (2) classify users based on common characteristics, such as their "role" ("is-a" class hierarchy); and (3) represent a set of users that is grouped specifically for the purposes of activating specific groupware tool(s). Note that we deliberately use the more general "group" term and avoid typical built-in mechanisms to define roles (which, as recognized by [14], can sometimes be as much limiting as assisting), although such a notion can be imposed on top of the generic user and group classes.

Once users and groups are modeled as objects, they can be pointed-to by other kinds of objects. For example, a multi-author document object can point to its "owner" object(s), and one user object can point to another "manager" user-object. This enables to query the user repository in conjunction with the artifact repository in order to, for example, automatically assign the review of a given document to its author(s) and notify their manager(s). But in order to provide such functionality, the repository must supply a mechanism that enables the process to select users based on the values of their attributes. The WFMS should strike the right balance between providing built-in support for few mandatory attributes of these classes, and allowing to extend these class definitions with optional, process-specific attributes that are defined and manipulated on a per-process basis.

To illustrate the concept of user modeling, consider the following sample *user* and *group* classes defined in the Oz data modeling language, shown in Figure 2. In this example, users are represented by the `USER` class. Each object that gets instantiated from that class has several "state" attributes (e.g., host from which user is connected, which is in general different from the host in which the `USER` object resides due to the client-server architecture, see Section 4), and a set of links to `USER_GROUP` instances. A WFMS supporting such class may elect to provide built-in support for none, some, or all of the attributes in that class. For example, when a user logs-in to the WFMS, the WFMS may automatically attach the user to his proper `USER`

```
USER_GROUP:: superclass PROTECTED_ENTITY;
        name : string;
        users: set_of link USER;
end
USER:: superclass PROTECTED_ENTITY;
        id :        userid;
        name:        string;
        host_name: string;
        host_ip:    string;
        active:     boolean;
groups:     set_of  link USER_GROUP;
end
```

Figure 2: **Sample User and Group Classes**

object by looking up the `userid` attribute, and subsequently fill-in some values for the built-in attributes based on the login information. The current version of Oz supports only the attachment to `USER` object as a built-in facility (actually, even this is not strictly enforced, since an Oz environment may have no user modeling at all in cases that it is unnecessary), although any other attribute can be defined and manipulated by a specific process.

3.2. User Binding

Once users and groups are properly modeled in the system, the process definition language must enable to associate user objects with the activities. This binding method depends on the underlying user-interaction and user-control models that WFMSs employ. In "active" WFMSs the process executes on behalf of the "system", in which case any (at least any interactive) activity must be assigned to some user, or to a set of users in case of a multi-user activity. In contrast, in "reactive", user-driven WFMSs, each activity is by default executed on behalf of the user who invoked it. In this case, single-user activities may still need to be *delegated* to other users. For example, ProcessWEAVER [8] supports *agendas*, which are personal "to do" list that can be updated by the WFMS or by other users for delegation purposes. Although delegation can be considered as a (somewhat restricted) form of groupware, we do not discuss it here; see [2].

The simplest method to bind users to activities is to specify them (via their object representation) directly in the process model as the "recipients" of the activity. The main problem with this static approach is that in order to change the binding set, the process model has to be modified and consequently the instantiated process must be evolved, a non-trivial task [19]. This is particularly evident in cases where the same activity could be bound to different users depending on the context in which the activity is invoked and other time- and location-dependent circumstances.

Our approach is to provide *dynamic* (late) user binding. When the process is initially defined, groupware activities are associated with *classes* of qualified users or groups, as opposed to instances. At run-time, they are attached to an activity based on both their class membership as well as the particular values of their state attributes. A group of participants can then be formed by either binding directly all (active) members of a given class, or by binding a set of individual users (which are not necessarily all part of the same group) which satisfy a certain condition.

By modeling users as objects and employing dynamic binding, a process can utilize the WFMS's regular data-query facilities in order to *select* the proper users based on the knowledge stored in these objects. For example, the process may be able to select only users that are known to be active, denoted by having a `true` value in the `active` attribute of their user object. Furthermore, if the process modeling language supports the notion of pre-condition or "guard" predicates (as many do), it can be applied to the user binding-set to check or verify that the selected user-set and its cardinality are valid for the activity. Finally, if the WFMS supports automatic invocation of activities to satisfy a failed condition (backward-activation), or following state-changes (forward-invocation), it can further assist in the process of locating qualified users. For example, a failed user-binding predicate could trigger an activity that can lookup, periodically poll, or otherwise employ a "wakeup" procedure to locate users.

Figure 3 shows an example of how the document-owner association mentioned earlier can be modeled in the Oz process modeling language (we ignore details of the language that are not relevant to the issue of user binding). The `multi-edit` activity takes a single document object as input (line 1). It then binds to the activity all users that satisfy two conditions (lines 4-7): they must be owners of the document (modeled as a link from the document to the owner object), and they must be active (as denoted by the `active` attribute). After binding other related documents (lines 9-10), the activity checks whether the users in the binding set are allowed to edit the document (line 14); if this is not the case, the activity is aborted. Otherwise, the user-binding phase actually takes place (denoted by the `participant` directive in line 16), followed by invocation of the multi-user tool (line 18) followed by assertions that reflect the result

```
1) multi-edit [?d:DOCUMENT]:
2) # OBJECT BINDING
3) (and
4)    # bind users to activity
5)    (forall USER ?u suchthat (and
6)                          (linkto [?doc.owner ?u])
7)                          (?u.active = true)))
8) # bind  relevant documents
9) (forall DOCUMENT ?related suchthat
10)                 (linkto [?doc.ref ?related]))):
11) # CONDITION
12) # check that documents are allowed to be read by all
13 ) # users in the binding set
14) (?doc.allowed_edit = ?u):
15) # USER BINDING
16) participants[?u]:
17) # EXECUTE
18)  MULTI_EDIT multi_editor ?doc ?related-docs
19) # ASSERT EFFECTS
20) #0. document changed.
21) #   this assertion may trigger other activities.
22) (?doc.status  = Changed);
23) #1. document unchanged.
24) (?doc.status  = NotChanged);
```

Figure 3: **multi-edit activity**

of applying the tool (lines 20-24). The actual interface from the modeling language to the tool is done via enveloping mechanism, which is beyond the scope of this paper, see [12].

Notice that depending on the selected document, each time the activity is invoked it would be assigned to the appropriate owners. Moreover, if the owners of the document change over time (this would be reflected by changing the `owner` links of the document), subsequent invocations of the same activity on the same document will automatically bind the new (active) owners.

Two additional closely-related issues to discuss regarding user-binding are: (1) "user-overflow", i.e., when the user binding-set contains more users than required by the activity; and (2) "user-underflow", when the user binding-set is smaller than required, either because there are no available users, or some users do not want to participate in such activity. In either case, some additional semantics must be associated, either by default and/or specified by additional syntax. Regarding overflow, the process modeling language needs to provide directives for subset-selection. Three plausible methods may be (1) interactive, i.e., the system prompts the coordinator who initiated the activity to make the selection; (2) random; and (3) following some priority scheme. The problem with user-underflow is more severe, since it effectively disables the execution of the activity, and therefore some failure semantics must be attached and further actions may be taken. We have identified the need for a `notify` action that directs the WFMS to notify (e.g., by e-mail) potential participants about future invocations, perhaps with a later `retry` action that actually re-invokes the activity. Alternatively, if the activity must be executed instantly, the WFMS may seek means to automatically activate users by invoking a triggering activity that would search for, and locate, qualified but currently inactive users.

3.3. Process Automation of Groupware Activities

Process automation lies in the heart of workflow management. Regarding automation groupware activities, we have already mentioned one form of automation, namely the ability to automatically satisfy a user-binding condition for setting-up a groupware activity. Another important capability that is required especially for groupware tools, is to be able to fan-in to and fan-out from the synchronous groupware activity and perform in parallel and asynchronously local and personal activities. This gives rise to distinguishing between globally defined, shared, multi-user activities, and locally defined, personalized activities, which each participant could define in his own private (sub-) process. In other words, if the process allows to define private workspaces with their own rules and state, the workflow engine could spawn multiple single-user activities as a result of, or as a preparation for, a groupware activity, without actually being required to know *how* these local activities are performed, and therefore delegating non-global definitions to the local users/sites. For example, in the Review task mentioned earlier, the `review` activity could be defined autonomously (and differently) by each user according to his own method of reviewing documents. This is the underlying motivation for the Summit model elaborated in [2] (although it actually deals with the general case of interoperating full-fledged processes). We will illustrate this control mechanism in Section 4.2.

3.4. Infrastructure Support

The above discussion made some implicit assumptions about the infrastructure support for groupware. This is in general open-ended and depends on the level of integration with the WFMS, but there are several basic requirements. First, the WFMS should be able to "locate" selected participants. This means that the system can locate and communicate with the client on whose behalf the participant executes (assuming a client-server WFMS as in the reference architecture [23]). Second, the WFMS should be able to (re)direct activities or parts of them across and among clients. Third, there must be a notification mechanism. Groupware activities, even if originating from one user, require to notify other remote users and ask them to perform the synchronous interaction. This setup procedure requires asynchronous mechanisms in which a client is notified and acts in reaction to a server request, which is the opposite of conventional client-server interaction. A related aspect is concerned with the user-interface for such notification. Such mechanism has to prompt the (perhaps unexpecting) user and attract its attention, and must allow the user the option to refuse to perform the action (or request to delay it).

4. Groupware Integration in Oz

We outline the implementation of the above ideas in the Oz framework and illustrate their use in an effective process for the Review task discussed in the introduction.

4.1. Oz Overview

Oz [2] is a multi-site collaborative WFMS that supports interoperability among heterogeneous and autonomous processes. Initially, it was designed to support software engineering projects (also known as process-centered software engineering environments, see [10] for a book surveying such systems), but later has been generalized to support workflow in various application domains (e.g., healthcare workflow [22]).

Oz introduces a flexible and dynamic mechanism to specify and integrate the desired interoperability between multiple process models (called the *Treaty* protocol), and corresponding execution support for multi-process activities that enables execution of activities with data/tools/users from multiple sites and fulfillment of their local prerequisites and consequences (the *Summit* protocol).

The architecture of Oz is illustrated in figure 4; it is "shared nothing", i.e., the system is physically as well as logically decentralized, with each site maintaining autonomously its own project database, schema, process, and tool base, thus not limiting a priori the scale, scope or physical location(s) of the project being developed. Interactions among sites utilize local client-remote server and server-to-server connectivity facilities provided by the system (local client-local server connections assume a shared network file system, but need not reside on the same host).

Treaties, Summits and interconnectivity support are discussed elsewhere [2, 4, 3]; here we investigate collaboration among multiple users regardless of whether they work within the same or in different processes, or within the same local area network vs. across a wide area network. Nevertheless, decentralization and geographical dispersion were prime motivations for this work: we discovered quickly the limitations of ad hoc approaches when we had to deal with interactions among logically and physically distributed users.

Human interaction with the environment is through a client that provides the user interface as well as the workspace in which activities are executed. When a user issues a command (often indicating a rule, see below) he/she wants to perform, the request is sent to the server to check whether the activity can be executed (e.g., all prerequisites are satisfied and no parameters are locked exclusively by another user) and explores opportunities to automate and/or guide the execution of this or related activities (e.g., invoking other activities to attempt to satisfy the prerequisites or negotiating according to relaxed concurrency control policies [15]), and eventually returns to the client — either with the necessary information to execute the activity or to inform the client that the activity cannot be performed at this time.

A local process in Oz is defined using a rule-based language. Each activity is enclosed in a rule with formal typed parameters, and optional condition and effects that serve two purposes: to enforce and assert conditions that pertain to the activity itself; and to connect (through predicate/assertion matching) to other related activities and specify automation and/or atomicity requirements across activities. Related activities can be invoked automatically as part of either backward chaining to satisfy the predicates in a rule's condition, or forward chaining as a result of the assertions in a rule's selected effect (a rule may define more than one effect, but exactly one is asserted depending on the results of the encapsulated activity). A rule thus defines a process step, and the set of all chains emanating from that rule (implicitly) define a task.

Oz allows for dynamic (or late) binding of data to activities. The actual parameters to the rule are designated either explicitly by the user, or automatically by the process engine in case of a chained-to rule (see [16]). In addition, the language binds derived parameters in a *binding clause* that queries the

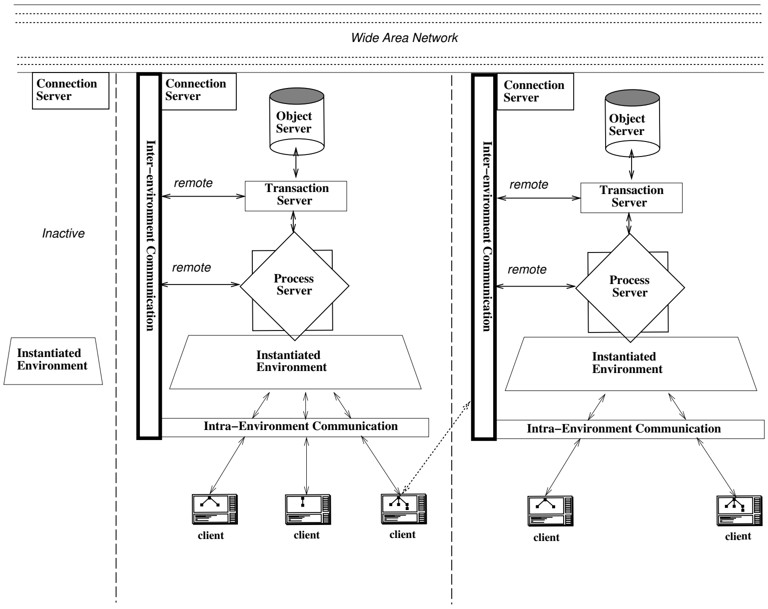

Figure 4: **Oz External Architecture**

project database, usually resulting in a set of objects structurally and/or logically related to the actual parameters.

Oz integrates most of the groupware facilities that were mentioned in the previous section, including user modeling, binding, automation, and infrastructure support for locating, setting-up, and notifying users through their clients. We now turn to the realization of the motivating example and show how the WFMS features can be exploited to support such process.

4.2. Realization of the Review Task

An Oz process that supports the Review task consists of: a schema (data model) with definitions for artifacts such as documents, reviews, revisions, etc.; a user model with proper `user` and `group` classes; envelopes encapsulating the tools, for example a conferencing tool [1]; and, most importantly from this paper's viewpoint, a set of rules that specify how and by who the tools are enacted, their inter-dependencies with other rules, and potential for their automatic invocation.

Each of the boxes in Figure 1 could be realized as either a single rule, or as a set of interrelated rules collectively implementing the (sub)task. The edges between the boxes are represented by the matchings between effects of one rule and the condition of another rule. A rule is designated as either a single-user personal rule or as a multi-user, groupware rule (for pragmatic reasons the actual annotation is made in the tool definition as opposed to in the enclosing rule , but this is an irrelevant syntactic detail). If a multi-user rule is encountered, the rule-processor employs its infrastructure to select, locate, and connect to the proper participants; set up the activation and transfer control for the execution of the external tool; and regain control after the termination of the activity, setting the proper state values and possibly invoking other single- or multi-user derived rules.

Figure 5 shows two sample rules from the process (a full realization of a comprehensive multi-user "benchmark" process appears in [2]): the groupware `conference` rule and the personal `review` rule that precedes it. The `conference` activity binds all members of a group that is linked to the document (lines 3-6, 12); it is enabled only if a conference is requested on the document (line 13); it invokes the `conference` groupware tool (line 15); and it asserts one of the three possible outcomes of the conference that correspond to the three boxes in Figure 1: revise, reject, or accept (lines 16-21). Depending on the outcome of the conference, the proper rule will be triggered and assigned to the proper user. The `review` rule has a pre-condition that states that the document must be

[1] In an actual implementation of a similar task we used the `white_board` public-domain tool, which enables multiple distributed users to share a virtual white-board on their screens [18].

```
1) conference[?d:DOCUMENT]:
2)  (and
3)   (forall USER_GROUP ?g      suchthat
4)                              (linkto [?d.reviewers ?g]))
5)   (forall USER        ?users  suchthat
6)                              (member [?g.users ?users]))
7)   (forall REVIEW      ?revs   suchthat
8)                              (linkto [?d.reviews ?revs]))
9)   (forall DOCUMENT    ?rel    suchthat
10)                             (linkto [?d.related ?rel])))
11)  :
12)  participants[?users]:
13)  (?revs.status = ConferenceRequested)
14)  # invoke the multi-user white_board tool
15)    MU_TOOLS conference ?d ?u ?revs ?rel
16)  # 0. enable revise
17)  (?d.review_status   = RevisionRequested);
18)  # 1. no hope, go to reject. needs to start all over again.
19)  (?d.review_status = Rejected));
20)  # 2. review accepted.
21)  (?d.review_status = Accepted);
###############
22)  review[?d:DOCUMENT, ?r:USER]:
23)  (exist REVIEW        ?review   suchthat
24)                      (and
25)                        (linkto [?d.reviews ?review])
26)                        (?review.owner = ?r.userid)))
27)    :
28)   (?review.status = ReviewRequested):
29)   delegate[?r]:
30)   REVIEW review ?d
31)   # 0. enable review
32)   (?review.status = ConferenceRequested);
33)   # 1. indicate error
34)   (?review.status = Error);
```

Figure 5: **Sample rules from the Review Task**

in a "reviewable" state (e.g. the author has completed it), and an assertion that either enables later invocation of `conference` (line 32), or disables it (line 34), depending on the outcome of the local `review` rule. (The `delegate` directive in line 29 is similar to the `participants` clause and binds the personal activity to a single user). Note that, as specified in the process definition, we want to enable `conference` only if and when `all` participants (i.e., reviewers) have completed their reviews. This is indicated by the `forall` universal quantifier (which is defined in line 7 but is actually used in line 13) that ensures the desired behavior (the latest version of Oz corrected this problem and the quantifiers are defined in the condition clause instead of the binding, as they should be).

5. Summary

The approach presented in this paper shows how synchronous groupware activities can be synergized with workflow technology in a way that exploits the advantages of both worlds. By integrating such tools into a process framework, we enable to apply on them all the advantages that workflow provides: (1) Formal definition in the context of an enclosing process, and in specifying constraints and guidelines for user binding and invocation in general; (2) assistance in the execution of such activities, by allowing the activities to modify the state of the process, thereby allowing to assist, automate, enforce, and monitor the activities as well as related (perhaps personal and asynchronous) activities.

The work described in this paper mainly deals with essentially one process state. Expanding groupware to work across multiple processes with their own process state and "user space", running on true heterogeneous WFMSs, is a major future direction of this work.

Acknowledgments

We would like to thank the members of the Programming Systems Laboratory at Columbia University. In particular, we would like to thank George Heineman and Peter Skopp for their overall contribution to the Oz project, and Giuseppe Valetto for his work on integrating asynchronous multi-user tools. We also thank Simon Kaplan, K. Narayanaswamy, and John Arnold for stimulating discussions on CSCW-process integration.

References

[1] Naser S. Barghouti and Gail E. Kaiser. Concurrency control in advanced database applications. *ACM Computing Surveys*, 23(3):269–317, September 1991.

[2] Israel Ben-Shaul and Gail E. Kaiser. *A Paradigm for Decentralized Process Modeling*. Kluwer Academic Publishers, Boston, 1995.

[3] Israel Z. Ben-Shaul and Gail E. Kaiser. A configuration process for a distributed software development environment. In *2nd International Workshop on Configurable Distributed Systems*, pages 123–134, Pittsburgh PA, March 1994.

[4] Israel Z. Ben-Shaul and Gail E. Kaiser. A paradigm for decentralized process modeling and its realization in the OZ environment. In *16th International Conference on Software Engineering*, pages 179–188, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[5] XSoft Marketing Communications. Inconcert: Workflow software from xsoft, October 1995. http://www.xerox.com/XSoft/DataSheets/InConcert.html.

[6] Prasun Dewan and John Riedl. Toward computer-supported concurrent software engineering. *Computer*, 26(1):17–36, January 1993.

[7] Michael E. Fagan. Advances in software inspections. *IEEE Transactions on Software Engineering*, SE-12(7):744–751, July 1986.

[8] Christer Fernström. PROCESS WEAVER: Adding process support to UNIX. In *2nd International Conference on the Software Process: Continuous Software Process Improvement*, pages 12–26, Berlin, Germany, February 1993. IEEE Computer Society Press.

[9] Rodrigo F. Flores. The value of a methodology for workflow, 1995. http://www.actiontech.com/market/papers/method5.html.

[10] Pankaj K. Garg and Mehdi Jazayeri, editors. *Process-Centered Software Engineering Environments*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[11] John Gintell, John Arnold, Michael Houde, Jacek Kruszelnicki, Roland McKenney, and Gerard Memmi. Scrutiny (TM): A collaborative inspection and review systems. In Ian Sommerville and Manfred Paul, editors, *4th European Software Engineering Conference*, number 717 in Lecture Notes in Computer Science, pages 344–360, Garmisch-Partenkirchen, Germany, September 1993. Springer-Verlag.

[12] Mark A. Gisi and Gail E. Kaiser. Extending a tool integration language. In Mark Dowson, editor, *1st International Conference on the Software Process: Manufacturing Complex Systems*, pages 218–227, Redondo Beach CA, October 1991. IEEE Computer Society Press.

[13] Jonathan Grudin. Computer supported cooperative work: History and focus. *Computer*, 27(5), May 1994.

[14] Dennis Heimbigner and Leon Osterweil. An argument for the elimination of roles. In Carlo Ghezzi, editor, *9th International Software Process Workshop*, Airlie VA, October 1994. IEEE Computer Society Press. In press.

[15] George T. Heineman and Gail E. Kaiser. An architecture for integrating concurrency control into environment frameworks. In *17th International Conference on Software Engineering*, Seattle WA, April 1995. In press.

[16] George T. Heineman, Gail E. Kaiser, Naser S. Barghouti, and Israel Z. Ben-Shaul. Rule chaining in MARVEL: Dynamic binding of parameters. *IEEE Expert*, 7(6):26–32, December 1992.

[17] D. Hollinsworth. The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition. http://www.aiai.ed.ac.uk/WfMC/.

[18] Van Jacobson and Steve McCanne. wb. van@ee.lbl.gov, mccanne@ee.lbl.gov, The development of 'wb', 'vat', and 'sd' were supported by the Director, Office of Energy Research, Scientific Computing Staff, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

[19] Gail E. Kaiser, Israel Z. Ben-Shaul, George T. Heineman, and Wilfredo Marrero. Process evolution for constraint-enforcing environments. Technical Report CUCS-047-92, Columbia University Department of Computer Science, March 1994.

[20] Simon M. Kaplan, William J. Tolone, Alan M. Carroll, Douglas P. Bogia, and Celsina Bignoli. Supporting collaborative software development with ConversationBuilder. In Herbert Weber, editor, *5th ACM SIGSOFT Symposium on Software Development Environments*, pages 11–20, Tyson's Corner VA, December 1992. Special issue of *Software Engineering Notes*, 17(5), December 1992.

[21] Kenyon Brown Kevin Brown and Kyle Brown. *Mastering LOTUS NOTES*. SYBEX, San Francisco CA, 1995.

[22] Wenke Lee, Gail E. Kaiser, Paul D. Clayton, and Eric H. Sherman. OzCare: A workflow automation system for care plans. Technical Report CUCS-012-96, Columbia University Department of Computer Science, March 1996. Submitted for publication.

[23] Workflow Management Coalition Members. Workflow management coalition reference model, February 1995. http://www.aiai.ed.ac.uk/WfMC/GIF/WfMC-ref-model.GIF.

[24] Giuseppe Valetto and Gail E. Kaiser. Enveloping sophisticated tools into process-centered environments. *Automated Software Engineering*. To appear.